CSC108: Introduction to Computer Programming

Lecture 10

Wael Aboulsaadat

Acknowledgment: these slides are based on material by: Velian Pandeliev, Diane Horton, Michael Samozi, Jennifer Campbell, and Paul Gries from CS UoT

Announcements

- Assignment 3 is due April 6, 2011….
- Since Quiz 3 was a bit too difficult for the time allotted, we are making it out of 8 instead of 10. This makes Quiz 3 average 60%.

Variable Scope (revisited)

Global Variables

- From what we know of namespaces, variables declared inside functions are destroyed after the function body finishes.

- If there is a variable that is useful to the entire module, it can be initialized outside of a function body and it will become part of the global namespace of the module.

- We call these variables **global**, **and they can be quite** useful, but there is a quirk of Python namespaces which requires us to be careful when using them.

Global Variables

- What will this code do?

```
glb = 17
def change_glb():
    glb = 3
def print_glb():
    print glb
change_glb()
print_glb()
```

It will print 17, not 3.

- Why?

Global Variables

- Functions should be encapsulated: they should work with the data they are given and return values as specified.
- It's generally not advisable for a function to change values outside its namespace, especially global values that someone else may want to use later.
- Therefore, by design in Python, any variable assigned to in a local namespace is assumed to be a local and is newly created in the local namespace.

Name spaces

- Python always looks for a name in the most local namespace first.

```
glb = 17
def change_glb():
    glb = 3
def print_glb():
    print glb
change_glb()
print_glb()
```

```
local change_glb():
    glb   3
```

```
global:
    glb   17
change_glb()-fxn
print_glb()-fxn
```

```
built-in:
abs    (function)
int    (function)
```

Global Variables

- Notice that accessing a global variable doesn't generate this pitfall.
- print_glb() works, retrieving and printing the global variable.
- The global variable is only superseded if its name is being assigned to in a local namespace.
- If a programmer is certain that (s)he wants to assign to a global variable, (s)he can declare this explicitly using the keyword global.

The Keyword global

- The keyword global placed before a name in a function specifies that that name should be looked for in the global namespace , not in that function's local namespace:

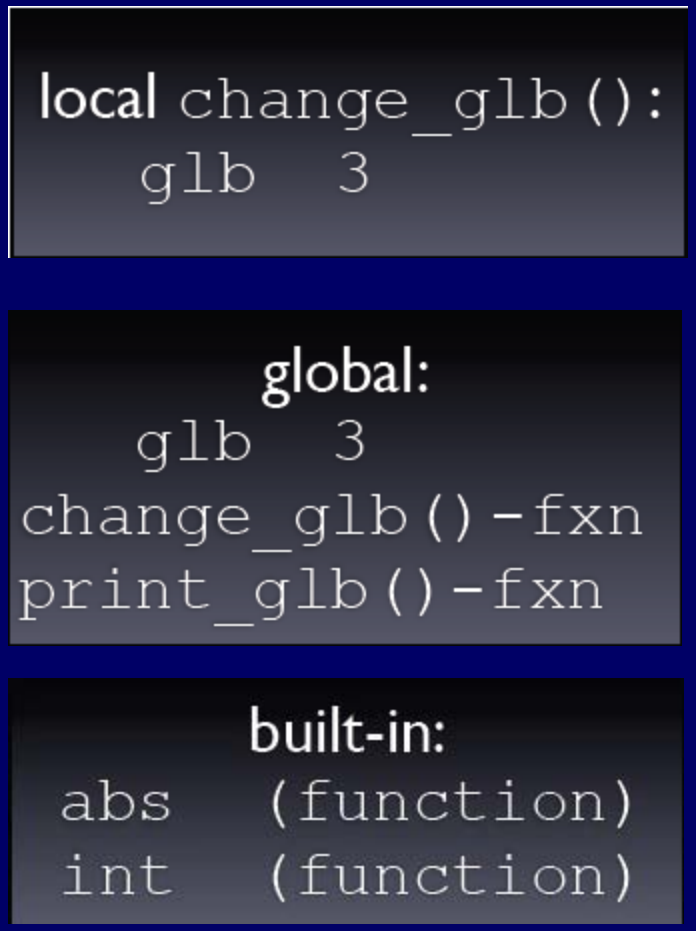
```
glb = 17
def change_glb():
        global glb
        glb = 3
def print_glb():
        print glb
change_glb()
print_glb()
```

- Will print 3, since global glb's value will be changed.

Namespaces

- A global declaration makes Python start looking in the global namespace.

```
glb = 17
def change_glb():
        global glb
        glb = 3
def print_glb():
        print glb
change_glb()
print_glb()
```



Functions & parameter passing (revisited)

Parameter Passing Rules

- Assuming we define a function as follows

```
def testParamPassing(X,Y,Z):
        print "X is", X
        print "Y is", Y
        print "Z is", Z
```

- What we have learnt is to call it as follows:

```
if __name__ == "__main__":
        testParamPassing(10,20,30)
```

Parameter Passing Rules

- Assuming we define a function as follows

```
def testParamPassing(X,Y,Z):
        print "X is", X
        print "Y is", Y
        print "Z is", Z
```

- What we have learnt is to call it as follows:

```
if __name__ == "__main__":
        testParamPassing(10,20,30)
```

- The interpreter supports a 1-to-1 mapping of parameters

Parameter Passing Rules

- Python supports another mechanism in passing parameters

```
def testParamPassing(X,Y,Z):
        print "X is", X
        print "Y is", Y
        print "Z is", Z
```

- We can name the parameter and assign it a value while calling a function:

```
if __name__ == "__main__":
        testParamPassing(X=10,Y=20,Z=30)
```

Parameter Passing Rules

- Python supports another mechanism in passing parameters

```
def testParamPassing(X,Y,Z):
        print "X is", X
        print "Y is", Y
        print "Z is", Z
```

- We can name the parameter and assign it a value while calling a function. The order does not matter anymore.

```
if __name__ == "__main__":
        testParamPassing(X=10,Y=20,Z=30)
        testParamPassing(Y=20,Z=30, X=10)
        testParamPassing(Z=30,X=10,Y=20)
```

Parameter Passing Rules – with defaults

- Recall that we can assign a default value to parameter

```
def testParamPassing(X,Y,Z=30):
        print "X is", X
        print "Y is", Y
        print "Z is", Z
```

- Assigning a default value to a parameter saves us from passing a value for that parameter

```
if __name__ == "__main__":
        testParamPassing(10,20)
```

Parameter Passing Rules – with defaults

- Recall that we can assign a default value to parameter

```
def testParamPassing(X,Y,Z=30):
        print "X is", X
        print "Y is", Y
        print "Z is", Z
```

- We can still using parameter names

```
if __name__ == "__main__":
        testParamPassing(X=10,Y=20)
        testParamPassing(Y=20, X=10)
```

Sorting (revisited)

Selection Sort

- In Selection Sort, we can think of our list as consisting of two parts: a sorted part and an unsorted part.

- Initially, the sorted part is empty, as presumably the entire list is unsorted. On every iteration, we:
  - find the smallest number in the unsorted part of the list
  - swap it with first number in the unsorted part of the list.
  - This increases the size of the sorted list by 1.
  - We repeat this process until the entire list is sorted.

Selection Sort Example

- Sorted part is yellow

Step 0: **[5, 3, 4, 7, 9, 2, 1]**

Step 1: **[1, 3, 4, 7, 9, 2, 5]**

Step 2: **[1, 2, 4, 7, 9, 3, 5]**

Step 3: **[1, 2, 3, 7, 9, 4, 5]**

Step 4: **[1, 2, 3, 4, 9, 7, 5]**

Step 5: **[1, 2, 3, 4, 5, 7, 9]**

Step 5: **[1, 2, 3, 4, 5, 7, 9]**

Step 6: **[1, 2, 3, 4, 5, 7, 9]**

Selection Sort Code

This Week's To Do List

- Go through lecture slides – make sure you try the code snippets
- Try the lecture's programs posted on course website